$1.80

Apple Assembly Line

Volume 5 -- Issue 2 November, 1984

In This Issue...

Apple II Troubleshooting Guide

We have just received a new book from Howard Sams: Apple II+/IIe Troubleshooting & Repair Guide, by Robert C. Brenner. At a glance, it looks like quite a good introduction to the Apple hardware and its potential problems. The first chapter is Basic Troubleshooting, followed by three chapters on Description, Operations, and Specific Troubleshooting for the II Plus, three more similar chapters on the //e, and two chapters on Preventive Maintenance and Advanced Trouble-shooting. Here's a quote from the Introduction:

> This book is a detailed troubleshooting and repair document. It is not a treatise on basic computer theory or a discussion of chip operation, registers, busses, and logic gates. It is an all "meat and potatoes" manual to enable the computer user to repair his or her own machine in those 95 percent of circumstances where knowledge and a good reference are enough to find and repair a failure.

List price of the Troubleshooting & Repair Guide is $19.95. Our price will be $18 + shipping.

18-Digit Arithmetic, Part 7...............Bob Sander-Cederlof

Last month we began the implementation of math functions, so it seems appropriate to continue in the same direction. This month we will reveal the LOG and EXP functions.

As always, I turned to "Computer Approximations" for some good algorithms. I mentioned this book last month, and several of you have tried to find copies.

Thanks to Trey Johnson, of Monolith Inc. in San Antonio, for the following information: John Wiley & Sons stopped publishing the book "Computer Approximations" in 1977. They sold the rights to Krieger Publishing Co., and it is now being published under the same title. Trey was quoted a price of $22.50 + shipping. Krieger's address is P. O. Box 9542, Melbourne, FL 32901; phone is (305) 724-9542.

"Computer Approximations" is the only book I have found which lists all the actual coefficients needed to produce good approximations for the whole variety of standard functions. Pages 189-339 are packed solid with nothing by numbers. For example, there are ten pages of numbers for the EXP function alone, providing over 100 different approximation formulas for the EXP function. The chapter covering EXP describes the math behind the approximations. You pick an algorithm according to the precision you need, the number base you are using (2, 10, or whatever), the tradeoff between speed and size, and the range of arguments you will be using. Each algorithm in the book has a number, and I indicate that number in the comments to the programs which follow.

Almost all of the approximations involve these steps:

- SIFT: Check the argument for legal range and easy arguments.
- FOLD: Reduce the range of the argument.
- POLY: Use a polynomial or a ratio of polynomials to approximate the function in the reduced range.
- UNFOLD: Expand the result by the reverse of the processes used to reduce the range.

When we first learned about logarithms in high school, we used tables in books. One set of tables converted normal numbers to logs, and the other converted logs back to normal numbers. The LOG function takes the place of the first set of tables, and the EXP function replaces the second. By the way, those high school logarithms were base 10 logs. The log of a number is the power to which you would have to raise 10 to equal the number. For example, the log base 10 of 1000 is 3; of the square root of 10 is .5.

Scientists prefer base "e" logs. "e" is an irrational number (as is pi) approximately equal to 2.7182818284590452353б. Did the original scientists have 2.718281828... fingers? Maybe, if they had to chop firewood (logs?)! Anyway, EXP and LOG in Applesoft work with base e. LOG tells you to what power you

S-C Macro Assembler Version 1.0..$80
S-C Macro Assembler Version 1.1.......................................$92.50
Version 1.1 Update..$12.50
Source Code for Version 1.1 (on two disk sides)........................$100
Full Screen Editor for S-C Macro (with complete source code)............$49
S-C Cross Reference Utility (without source code).......................$20
S-C Cross Reference Utility (with complete source code).................$50
DISASM Dis-Assembler (RAK-Ware)...$30
Source Code for DISASM.......................................additional $30

S-C Word Processor (with complete source code)..........................$50
Double Precision Floating Point for Applesoft (with source code)........$50
S-C Documentor (complete commented source code of Applesoft ROMs).......$50
Source Code of //e CX & F8 ROMs on disk.................................$15

(All source code is formatted for S-C Macro Assembler Version 1.1. Other assemblers require some effort to convert file type and edit directives.)

AAL Quarterly Disks...each $15

Each disk contains all the source code from three issues of "Apple Assembly Line", to save you lots of typing and testing time.

QD#1: Oct-Dec 1980	QD#2: Jan-Mar 1981	QD#3: Apr-Jun 1981
QD#4: Jul-Sep 1981	QD#5: Oct-Dec 1981	QD#6: Jan-Mar 1982
QD#7: Apr-Jun 1982	QD#8: Jul-Sep 1982	QD#9: Oct-Dec 1982
QD#10: Jan-Mar 1983	QD#11: Apr-Jun 1983	QD#12: Jul-Sep 1983
QD#13: Oct-Dec 1983	QD#14: Jan-Mar 1984	QD#15: Apr-Jun 1984
QD#16: Jul-Sep 1984		

AWIIe Toolkit (Don Lancaster, Synergetics)..............................$39
Quick-Trace (Anthro-Digital)...........CLOSEOUT SPECIAL!..(reg. $50) ~~$45~~ $35
Visible Computer: 6502 (Software Masters)................(reg. $50) $45
ES-CAPE: Extended S-C Applesoft Program Editor..........................$60
"Bag of Tricks", Worth & Lechner, with diskette.............($39.95) $36

Blank Diskettes (Verbatim)............2.25 each, or package of 20 for $40
(Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
or $25 per 100
These are cardboard folders designed to fit into 6"X9" Envelopes.
Envelopes for Diskette Mailers............................. 6 cents each
QuikLoader EPROM System (SCRG)................................($179) $170
D MAnual Controller (SCRG).....................................($90) $85
Switch-a-Slot (SCRG)..($190) $175
Extend-a-Slot (SCRG)...($35) $32

Books, Books, Books.........................compare our discount prices!

"Apple II+/IIe Troubleshooting & Repair Guide", Brenner.($19.95) $18
"Apple][Circuit Description", Gayler..................($22.95) $21
"Understanding the Apple II", Sather....................($22.95) $21
"Enhancing Your Apple II, vol. 1", Lancaster............($15.95) $15
Second edition, with //e information.
"Assembly Cookbook for the Apple II/IIe", Lancaster.....($21.95) $20
"Incredible Secret Money Machine", Lancaster.............($7.95) $7
"Beneath Apple DOS", Worth & Lechner....................($19.95) $18
"Beneath Apple ProDOS", Worth & Lechner.................($19.95) $18
"What's Where in the Apple", Second Edition.............($19.95) $19
"6502 Assembly Language Programming", Leventhal.........($18.95) $18
"6502 Subroutines", Leventhal...........................($18.95) $18
"Real Time Programming -- Neglected Topics", Foster......($9.95) $9
"Microcomputer Graphics", Myers.........................($12.95) $12

We have small quantities of other great books, call for titles & prices. Add $1.50 per book for US shipping. Foreign orders add postage needed.

Texas residents please add 6 1/8 % sales tax to all orders.

*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
*** (214) 324-2050 ***
*** We accept Master Card, VISA and American Express ***

would raise e to equal the argument, and EXP raises e to the power of the argument.

One great application of LOG and EXP is to raise any number to any power. Applesoft (as well as DP18) has an exponentiation operator "^" for this purpose, but the code inside does it by calling on EXP and LOG. Here are some mathematical symbols to indicate how it is done:

```
 let          z = x^y
then     log z = log (x^y)
         log z = y log x
   exp (log z) = exp (y log x)
           x^y = exp (y log x)
```

Here is the code for the exponentiation operator in DP18:

```
*--------------------------------------
*   EXPONENTIATION:  X ^ Y
*       (DAC) = Y
*       (ARG) = X
*--------------------------------------
DP.POWER
       JSR MOVE.DAC.TEMP3   SAVE DAC (POWER) IN TEMP3
       JSR SWAP.ARG.DAC
       JSR DP.LOG10         GET LOG X
       JSR MOVE.TEMP3.ARG   GET Y IN ARG
       JSR DMULT            Y LOG X
       JMP DP.EXP10         X ^ Y
```

Notice I used base 10 log and exp? That is because DP18 is basically decimal. In a binary floating point scheme such as is internal to Applesoft, base 2 log and exp would probably be used. After all, floating point notation is a kind of half-log half-normal notation.

Which leads to the topic of converting from one logarithmic base to another. If my internal subroutines work in base 10, how do I get LOG and EXP to base e? Some more math is due:

```
suppose          e^x = 10^y
then     log10 (e^x) = log10 (10^y)
          x log10(e) = y log10(10)
          x log10(e) = y
```

Log10(e) is a constant, approximately 0.4342944819032518276S. So if I want to know what EXP(3) is, I can first get 3*log10(e) = 1.302..., and 10^1.302... = 20.0855...

EXP Function

Lines 1640-1660 of the program check for a zero argument, which is an easy case: e^0 = 1. Lines 1670-1700 multiply the argument by log10(e), so that EXP10 can be used.

Lines 1730-1740 again sift out the easy case of 10^0, in case DP.EXP10 was called directly.

Lines 1750-1790 begin the folding process. We can cut the range in half by folding all negative arguments on top to the positive range: EXP(-x) = 1/EXP(x).

Lines 1810,1820 further sift, by eliminating arguments larger than 99. If the exponent of the argument is $43 or more, then the argument is 100 or more. Arguments that large are too large. (Indeed, any argument above 63 is too large.) The Applesoft ROM routine for OVERFLOW ERROR will let you know you tried it.

The arguments we have left will be in the range 0 < x < 100. We can further subdivide the range by separating the integer and fractional parts of the argument. Remember that 10^(x+y) = (10^x)*(10^y)? For illustration, suppose the argument is 3.75. Then 10^3.75 = 10^3 * 10^.75 = 5623.4132.... Lines 1830-2100 perform the separation. The variable INTPWR will get the integer part, which may range from 0 to 99. The corresponding digits are zeroed in DAC, and the resulting fraction is re-normalized. If the fractional part is zero, then the log of the fractional part is 1; lines 2080-2100 sift out this special case. This section could be accomplished by using previously covered subroutines, such as DP.INT to get the integer part, and DSUB to get the fractional part. However, that would take considerably longer for only a slight savings in space.

The active part of the argument has now been reduced to the range 0<x<1. The next adjustment will cut that in half. If the argument x<.5, this adjustment will be skipped. Lines 2120-2160 perform the test, and line 2170 saves the result of the test on the stack. We need the result later when we are unfolding. If x >= .5, then lines 2190-2210 subtract .5 from it. If x = .5, then the result after subtraction will be zero. In this case, the correct answer is a known constant, the square root of 10. Lines 2230-2270 load up that value and skip over the POLY part on down to the UNFOLDing. If not exactly .5, we now have a folded argument in the range 0<x<.5, with a flag on the stack indicating whether or not we subtracted .5 to get there. Later, if we DID subtract .5, we will multiply the result of POLY by the square root of 10 to unfold the answer.

We could have arbitrarily subtracted .5, changing the range from 0<x<1 to -.5<x<.5, with the same result. This would have saved the trouble of determining which side of .5 we were on, and of later deciding whether or not to multiply by SQR(10). However, it would also take longer for those cases already under .5, so I decided against it.

The POLY part is lines 2280-2520. This is a ratio of two polynomials, both 8th degree. However, because of derivational and computational reasons, it is actually written and calculated in a different form:

```
                Q(x^2) + xP(x^2)
      POLY(x) = ----------------
                Q(x^2) - xP(x^2)
```

Lines 2290-2320 save x and compute x^2. Lines 2330-2380 call on POLY.N (covered last month) to compute the P polynomial, and then multiply the result by x. The constants are given in lines 1440-1490. So that you see the form, I will give it here with the coefficients rounded off:

$$xP = 31x^7 + 4562x^5 + 134331x^3 + 760254x$$

Lines 2400-2430 compute the Q-polynomial, by calling POLY.1 (also covered last month). POLY.1 is used when the coefficient of the highest degreed term is 1. We get, approximately,

$$Q = x^8 + 477x^6 + 29732x^4 + 408437x^2 + 660349$$

Lines 2440-2520 form the numerator and denominator and divide, giving us a very nice approximation to the function for the folded argument.

Lines 2530-2590 begin the unfolding process, by multiplying by SQR(10) if we previously folded .5<x<1 down to 0<x<.5.

Lines 2600-2660 take care of the integral portion of the original argument, by adding it to the EXPONENT of the result so far. This is equivalent to multiplying by the integral power of ten, but much faster. Isn't base ten nice?

The final adjustment is to take the reciprocal if the original argument was negative, done in lines 2670-2730.

LOG Function

The LOG function is the inverse of the EXP function. Now if we could just run the 6502 backwards....

Log base e is related to log base 10 the same way the exp functions were:

$$\log_e x = \log_e(10) * \log_{10}(x)$$

Lines 2990-3040 call on the LOG10 subroutine and then multiply the result by the log base e of 10.

The LOG10 routine begins by sifting out the objectionable argument values, at lines 3100-3130. The argument MUST be positive, and MUST NOT be zero. Negative or zero arguments send you to Applesoft's ILLEGAL QUANTITY ERROR.

Lines 3140-3170 separate the exponent from the mantissa of the argument. The exponent represents the power of 10 multiplier, so as an integer it can just be added to the logarithm of the mantissa viewed as a fraction. The exponent is saved in INTPWR, to be processed later. Stuffing $40 in its place in DAC makes the range now .1<=x<1.

Lines 3180-3210 multiply the fraction by SQR(10), which changes the range to

$$\frac{1}{SQR(10)} \text{ <= } x < SQR(10)$$

This can be compensated for later by subtracting .5 from the logarithm of the folded argument.

Lines 3220 further thrash the argument by forming an intermediate argument z = (x-1)/(x+1). This value z will be in the range -.52 < z < +.52, which is a nice symmetrical value to run through a ratio of polynomials. I get lost in the math that motivates this step.

The POLY part is again a ratio of two polynomials. Lines 3330-3440 calculate the numerator, which is approximately

    -15z^11 + 301z^9 - 1726z^7 + 4060z^5 - 4192z^3 + 1576z

The denominator, formed in lines 3450-3500, is approximately

    z^12-68z^10+764z^8-3200z^6+6122z^4-5432z^2+1815

Dividing at line 3510 gives the logarithm of the value x. To unfold, we need to subtract .5, handled by lines 3860-3920. We also need to add as an integer the power of ten we saved in INTPWR. The latter is trickier, because we must convert a biased binary integer to a signed decimal floating point value.

Lines 3530-3600 un-bias INTPWR. If the exponent happens to be exactly $40, which in un-biased terms is 0, the rest of this step can be skipped (because the log of 10^0 is zero, adding nothing). If not, it is time to build a DP18 value in ARG. Line 3570 saves the sign in ARG.SIGN.

Lines 3610-3620 pre-clear ARG.HI, which is where we will be putting the one or two digits of INTPWR. Line 3630 assumes it will be a one-digit value, and lines 3640-3650 test that assumption. If it is one digit, lines 3730-3780 will shift the digit to the left nybble and store it in ARG.HI. If two digits, lines 3660 will divide by ten to get the high digit as quotient and low digit as remainder. Then lines 3730-3780 will merge the two digits into ARG.HI.

Lines 3790-3840 complete the construction of ARG by storing the exponent and clearing the remaining mantissa bytes. Line 3850 adds the value to the results of the POLY step, lines 3870-3920 subtract .5, and the answer is ready.

```
                  1000 *SAVE S.DP18 FUNC LOG
                  1010 *--------------------------------
E8D5-             1020 AS.OVRFLW   .EQ $E8D5
E199-             1030 AS.ILLERR   .EQ $E199
                  1040 *--------------------------------
FFFF-             1050 POLY.1      .EQ $FFFF
FFFF-             1060 POLY.N      .EQ $FFFF
FFFF-             1070 DADD        .EQ $FFFF
FFFF-             1080 DSUB        .EQ $FFFF
FFFF-             1090 DMULT       .EQ $FFFF
FFFF-             1100 DDIV        .EQ $FFFF
FFFF-             1110 DP.TRUE     .EQ $FFFF
FFFF-             1120 MOVE.YA.ARG.1  .EQ $FFFF
FFFF-             1130 MOVE.YA.DAC.1       .EQ $FFFF
FFFF-             1140 SWAP.DAC.ARG   .EQ $FFFF
```

```
FFFF-              1150 MOVE.TEMP1.ARG      .EQ $FFFF
FFFF-              1160 MOVE.TEMP2.ARG      .EQ $FFFF
FFFF-              1170 MOVE.TEMP3.ARG      .EQ $FFFF
FFFF-              1180 MOVE.DAC.ARG        .EQ $FFFF
FFFF-              1190 MOVE.TEMP3.DAC      .EQ $FFFF
FFFF-              1200 MOVE.DAC.TEMP1      .EQ $FFFF
FFFF-              1210 MOVE.DAC.TEMP2      .EQ $FFFF
FFFF-              1220 MOVE.DAC.TEMP3      .EQ $FFFF
FFFF-              1230 NORMALIZE.DAC       .EQ $FFFF
                   1240 *--------------------------------
0800-              1250 DAC.EXPONENT .BS 1
0801-              1260 DAC.HI       .BS 10
080B-              1270 DAC.SIGN     .BS 1
                   1280 *--------------------------------
080C-              1290 ARG.EXPONENT .BS 1
080D-              1300 ARG.HI       .BS 10
0817-              1310 ARG.SIGN     .BS 1
                   1320 *--------------------------------
0818-              1330 SIGN         .BS 1
0819-              1340 INTPWR       .BS 1
                   1350 *--------------------------------
081A- 41 10 00
081D- 00 00 00
0820- 00 00 00
0823- 00 00        1360 CON.ONE    .HS 41.10000.00000.00000.00000
0825- 40 50 00
0828- 00 00 00
082B- 00 00 00
082E- 00 00        1370 CON.1HALF  .HS 40.50000.00000.00000.00000
0830- 41 31 62
0833- 27 76 60
0836- 16 83 79
0839- 33 20        1380 CON.SQR10  .HS 41.31622.77660.16837.93320
                   1390 *--------------------------------
                   1400 *      EXP (DAC)    E^DAC
                   1410 *             OR    10^DAC
                   1420 *      #1446 IN HART, ET AL
                   1430 *--------------------------------
083B-              1440 P.EXP    .EQ *
03-                1450 P.EXP.N  .EQ 3
083B- 42 31 34
083E- 11 79 40
0841- 19 73 04
0844- 87 77        1460          .HS 42.31341.17940.19730.48777
0846- 44 45 61
0849- 82 83 16
084C- 94 65 63
084F- 58 48        1470          .HS 44.45618.28316.94656.35848
0851- 46 13 43
0854- 31 13 47
0857- 35 85 55
085A- 90 34        1480          .HS 46.13433.11347.35855.59034
085C- 46 76 02
085F- 54 47 94
0862- 41 26 53
0865- 94 34        1490          .HS 46.76025.44794.41265.39434
0867-              1500 Q.EXP    .EQ *
04-                1510 Q.EXP.N  .EQ 4
0867- 43 47 70
086A- 54 40 30
086D- 08 20 79
0870- 87 75        1520          .HS 43.47705.44030.08207.98775
0872- 45 29 73
0875- 26 06 55
0878- 85 99 68
087B- 33 03        1530          .HS 45.29732.60655.85996.83303
087D- 46 40 84
0880- 36 97 96
0883- 67 73 62
0886- 82 36        1540          .HS 46.40843.69796.67736.28236
0888- 46 66 03
088B- 48 65 05
088E- 27 14 15
0891- 44 91        1550          .HS 46.66034.86505.27141.54491
                   1560 *--------------------------------
0893- 40 43 42
0896- 94 48 19
0899- 03 25 18
089C- 27 65        1570 CON.LOGE .HS 40.43429.44819.03251.82765
```

```
                  *-------------------------------
                  DP.EXP.NULL
089E- 4C FF FF           JMP DP.TRUE  E^0 = 10^0 = 1.0
                  DP.EXP.OVERFLOW
08A1- 4C D5 E8           JMP AS.OVRFLW
                  *-------------------------------
                  DP.EXPE
08A4- AD 00 08           LDA DAC.EXPONENT
08A7- F0 F5              BEQ DP.EXP.NULL
08A9- A9 93              LDA #CON.LOGE
08AB- A0 08              LDY /CON.LOGE
08AD- 20 FF FF           JSR MOVE.YA.ARG.1
08B0- 20 FF FF           JSR DMULT    CHANGE TO 10^X
                  *-------------------------------
                  DP.EXP10
08B3- AE 00 08           LDX DAC.EXPONENT       10^0 = 1
08B6- F0 E6              BEQ DP.EXP.NULL
                  *---HANDLE NEGATIVE POWERS-------
08B8- AD 0B 08           LDA DAC.SIGN SAVE FOR 1/EXP IF NEGATIVE
08BB- 8D 18 08           STA SIGN
08BE- A9 00              LDA #0        GET ABS(X)
08C0- 8D 0B 08           STA DAC.SIGN
                  *---SPLIT INTEGER & FRACTION-----
08C3- E0 43              CPX #$43      THREE OR MORE INTEGER DIGITS?
08C5- B0 DA              BCS DP.EXP.OVERFLOW   YES, OVERFLOW
08C7- A9 00              LDA #0        ...ALL FRACTIONAL
08C9- 8D 19 08           STA INTPWR
08CC- E0 41              CPX #$41
08CE- 90 35              BCC .3        ...NO INTEGRAL PART
08D0- AD 01 08           LDA DAC.HI    ...1 OR 2 DIGITS
08D3- 4A                 LSR
08D4- 4A                 LSR
08D5- 4A                 LSR
08D6- 4A                 LSR
08D7- 8D 19 08           STA INTPWR
08DA- AD 01 08           LDA DAC.HI
08DD- 29 0F              AND #$0F
08DF- 8D 01 08           STA DAC.HI
08E2- E0 41              CPX #$41      ONE OR TWO DIGITS?
08E4- F0 14              BEQ .2        ...ONE DIGIT INTEGER
08E6- AD 19 08           LDA INTPWR     DIGIT*10
08E9- 0A                 ASL
08EA- 0A                 ASL
08EB- 6D 19 08           ADC INTPWR
08EE- 0A                 ASL
08EF- 6D 01 08           ADC DAC.HI
08F2- 8D 19 08           STA INTPWR
08F5- A2 00              LDX #0
08F7- 8E 01 08           STX DAC.HI
08FA- 20 FF FF    .2     JSR NORMALIZE.DAC     ADJUST REMAINING FRACTION
08FD- D0 06              BNE .3                FRACTION NOT 0
08FF- 20 FF FF           JSR DP.TRUE           10^0 = 1
0902- 4C 78 09           JMP .7
                  *---ADJUST FRACTION SO < .5------
0905- AD 00 08    .3     LDA DAC.EXPONENT
0908- C9 40              CMP #$40
090A- 90 05              BCC .4
090C- AD 01 08           LDA DAC.HI
090F- C9 50              CMP #$50
0911- 08          .4     PHP           REMEMBER...
0912- 90 15              BCC .5        ...ALREADY < .5
0914- E9 50              SBC #$50
0916- 8D 01 08           STA DAC.HI
0919- 20 FF FF           JSR NORMALIZE.DAC
091C- D0 0B              BNE .5        ...REST OF FRACTION NOT 0
091E- 68                 PLA           POP SAVED STATUS
091F- A9 30              LDA #CON.SQR10
0921- A0 08              LDY /CON.SQR10
0923- 20 FF FF           JSR MOVE.YA.DAC.1
0926- 4C 78 09           JMP .7
                  *---COMPUTE 10^.XXXX------------
0929- 20 FF FF    .5     JSR MOVE.DAC.TEMP1    SAVE X
092C- 20 FF FF           JSR MOVE.DAC.ARG
092F- 20 FF FF           JSR DMULT             GET X^2
0932- 20 FF FF           JSR MOVE.DAC.TEMP2    SAVE X^2
0935- A9 3B              LDA #P.EXP            COMPUTE P(X^2)
0937- A0 08              LDY /P.EXP
0939- A2 03              LDX #P.EXP.N
```

```
093B- 20 FF FF 2360        JSR POLY.N
093E- 20 FF FF 2370        JSR MOVE.TEMP1.ARG    COMPUTE XP(X^2)
0941- 20 FF FF 2380        JSR DMULT
0944- 20 FF FF 2390        JSR MOVE.DAC.TEMP3    SAVE XP(X^2)
0947- A9 67    2400        LDA #Q.EXP            COMPUTE Q(X^2)
0949- A0 08    2410        LDY /Q.EXP
094B- A2 04    2420        LDX #Q.EXP.N
094D- 20 FF FF 2430        JSR POLY.1
0950- 20 FF FF 2440        JSR MOVE.DAC.TEMP2    SAVE Q(X^2)
0953- 20 FF FF 2450        JSR MOVE.TEMP3.ARG    NUMERATOR = Q+XP
0956- 20 FF FF 2460        JSR DADD              Q(X^2)+XP(X^2)
0959- 20 FF FF 2470        JSR MOVE.DAC.TEMP1    SAVE UMERATOR
095C- 20 FF FF 2480        JSR MOVE.TEMP2.ARG    DENOMINATOR = Q-XP
095F- 20 FF FF 2490        JSR MOVE.TEMP3.DAC
0962- 20 FF FF 2500        JSR DSUB              Q(X^2)-XP(X^2)
0965- 20 FF FF 2510        JSR MOVE.TEMP1.ARG    10^.XXX = N/D
0968- 20 FF FF 2520        JSR DDIV
               2530 *---ADJUST BY SQR(10)------------
096B- 28       2540        PLP          SEE IF ADJUSTMENT NEEDED
096C- 90 0A    2550        BCC .7       ...NO
096E- A9 30    2560        LDA #CON.SQR10
0970- A0 08    2570        LDY /CON.SQR10
0972- 20 FF FF 2580        JSR MOVE.YA.ARG.1
0975- 20 FF FF 2590        JSR DMULT
               2600 *---ADD INTEGRAL POWER-----------
0978- 18       2610 .7     CLC
0979- AD 00 08 2620        LDA DAC.EXPONENT
097C- 6D 19 08 2630        ADC INTPWR
097F- 10 03    2640        BPL .8       ...NO OVERFLOW
0981- 4C A1 08 2650        JMP DP.EXP.OVERFLOW
0984- 8D 00 08 2660 .8     STA DAC.EXPONENT
               2670 *---ADJUST FOR SIGN--------------
0987- AD 18 08 2680        LDA SIGN     GET ORIGINAL SIGN
098A- 10 0A    2690        BPL .9       POSITIVE, WE ARE DONE
098C- A9 1A    2700        LDA #CON.ONE NEGATIVE, FORM RECIPROCAL
098E- A0 08    2710        LDY /CON.ONE
0990- 20 FF FF 2720        JSR MOVE.YA.ARG.1
0993- 20 FF FF 2730        JSR DDIV
0996- 60       2740 .9     RTS
               2750 *--------------------------------
               2760 *      LN (DAC)  LOG E (DAC)
               2770 *          OR   LOG 10 (DAC)
               2780 *      #2330 IN HART, ET AL
               2790 *--------------------------------
0997-          2800 P.LOG     .EQ *
05-            2810 P.LOG.N   .EQ 5
0997- C2 14 93
099A- 34 18 71
099D- 23 10 14
09A0- 98 68    2820           .HS C2.14933.41871.23101.49868
09A2- 43 30 13
09A5- 23 47 34
09A8- 14 74 84
09AB- 61 38    2830           .HS 43.30132.34734.14748.46138
09AD- C4 17 25
09B0- 53 62 65
09B3- 00 65 30
09B6- 33 87    2840           .HS C4.17255.36265.00653.03387
09B8- 44 40 59
09BB- 83 31 23
09BE- 94 47 62
09C1- 15 13    2850           .HS 44.40598.33123.94476.21513
09C3- C4 41 92
09C6- 34 56 02
09C9- 07 08 10
09CC- 79 11    2860           .HS C4.41923.45602.07081.07911
09CE- 44 15 76
09D1- 43 34 84
09D4- 51 12 76
09D7- 92 55    2870           .HS 44.15764.33484.51127.69255
09D9-          2880 Q.LOG     .EQ *
06-            2890 Q.LOG.N   .EQ 6
09D9- C2 67 69
09DC- 64 11 90
09DF- 46 22 45
09E2- 27 58    2900           .HS C2.67696.41190.46224.52758
```

```
09E4- 43 76 35
09E7- 70 02 30
09EA- 09 15 57
09ED- 98 77     2910        .HS 43.76357.00230.09155.79877
09EF- C4 32 00
09F2- 08 79 86
09F5- 36 66 41
09F8- 22 25     2920        .HS C4.32000.87986.36664.12225
09FA- 44 61 21
09FD- 60 00 41
0A00- 77 46 87
0A03- 80 69     2930        .HS 44.61216.00041.77468.78069
0A05- C4 54 31
0A08- 59 49 50
0A0B- 92 57 52
0A0E- 57 35     2940        .HS C4.54315.94950.92575.25735
0A10- 44 18 14
0A13- 93 61 20
0A16- 76 61 63
0A19- 02 82     2950        .HS 44.18149.36120.76616.30282
                2960 *--------------------------------
0A1B- 41 23 02
0A1E- 58 50 92
0A21- 99 40 45
0A24- 68 40     2970 CON.LN10 .HS 41.23025.85092.99404.56840
                2980 *--------------------------------
                2990 DP.LOGE
0A26- 20 36 0A  3000        JSR DP.LOG10
0A29- A9 1B     3010        LDA #CON.LN10     CONVERT LOG10 TO LN
0A2B- A0 0A     3020        LDY /CON.LN10
0A2D- 20 FF FF  3030        JSR MOVE.YA.ARG.1
0A30- 4C FF FF  3040        JMP DMULT
                3050 *--------------------------------
                3060 DP.LOG.ERR
0A33- 4C 99 E1  3070        JMP AS.ILLERR
                3080 *--------------------------------
                3090 DP.LOG10
0A36- AD 0B 08  3100        LDA DAC.SIGN      CHECK RANGE
0A39- 30 F8     3110        BMI DP.LOG.ERR    ...NEGATIVE
0A3B- AD 00 08  3120        LDA DAC.EXPONENT
0A3E- F0 F3     3130        BEQ DP.LOG.ERR    ...ZERO
0A40- 8D 19 08  3140        STA INTPWR        SAVE POWER OF 10
                3150 *---ADJUST RANGE-----------------
0A43- A9 40     3160        LDA #$40          MAKE FRACTION .1 TO .9999
0A45- 8D 00 08  3170        STA DAC.EXPONENT
0A48- A9 30     3180        LDA #CON.SQR10    1/SQR(10) ... SQR(10)
0A4A- A0 08     3190        LDY /CON.SQR10
0A4C- 20 FF FF  3200        JSR MOVE.YA.ARG.1
0A4F- 20 FF FF  3210        JSR DMULT
                3220 *---FORM (X-1)/(X+1)-------------
0A52- 20 FF FF  3230        JSR MOVE.DAC.TEMP1
0A55- 20 FF FF  3240        JSR MOVE.DAC.ARG
0A58- 20 FF FF  3250        JSR DP.TRUE       GET 1 IN DAC
0A5B- 20 FF FF  3260        JSR DSUB          X-1
0A5E- 20 FF FF  3270        JSR MOVE.DAC.TEMP2 SAVE IT
0A61- 20 FF FF  3280        JSR DP.TRUE       GET 1 IN DAC
0A64- 20 FF FF  3290        JSR MOVE.TEMP1.ARG
0A67- 20 FF FF  3300        JSR DADD          X+1
0A6A- 20 FF FF  3310        JSR MOVE.TEMP2.ARG
0A6D- 20 FF FF  3320        JSR DDIV          X-1/X+1
                3330 *---NUMERATOR = Z*P(Z^2)---------
0A70- 20 FF FF  3340        JSR MOVE.DAC.TEMP1 SAVE IT
0A73- 20 FF FF  3350        JSR MOVE.DAC.ARG
0A76- 20 FF FF  3360        JSR DMULT         Z^2
0A79- 20 FF FF  3370        JSR MOVE.DAC.TEMP2 SAVE Z^2
0A7C- A9 97     3380        LDA #P.LOG
0A7E- A0 09     3390        LDY /P.LOG
0A80- A2 05     3400        LDX #P.LOG.N
0A82- 20 FF FF  3410        JSR POLY.N
0A85- 20 FF FF  3420        JSR MOVE.TEMP1.ARG
0A88- 20 FF FF  3430        JSR DMULT         Z*P(Z^2)
0A8B- 20 FF FF  3440        JSR MOVE.DAC.TEMP1
                3450 *---DENOMINATOR = Q(Z^2)---------
0A8E- A9 D9     3460        LDA #Q.LOG
0A90- A0 09     3470        LDY /Q.LOG
0A92- A2 06     3480        LDX #Q.LOG.N
```

```
0A94- 20 FF FF          JSR POLY.1
0A97- 20 FF FF          JSR MOVE.TEMP1.ARG
0A9A- 20 FF FF          JSR DDIV            Z*P(Z^2)/Q(Z^2)
                *---ADD INTEGER POWER-----------
0A9D- 38                SEC
0A9E- AD 19 08          LDA INTPWR          GET POWER OF 10
0AA1- E9 40             SBC #$40
0AA3- F0 39             BEQ .5              ...0, NO NEED TO ADD ANYTHING
0AA5- 8D 17 08          STA ARG.SIGN
0AA8- B0 04             BCS .1              ...1 TO 63
0AAA- 49 FF             EOR #$FF            MAKE IT POSITIVE
0AAC- 69 01             ADC #1
0AAE- A0 00     .1      LDY #0
0AB0- 8C 0D 08          STY ARG.HI
0AB3- A2 41             LDX #$41
0AB5- C9 0A             CMP #10
0AB7- 90 0B             BCC .3              1...9
0AB9- E8                INX                 10...63
0ABA- 8D 0D 08  .2      STA ARG.HI          STORE REMAINDER
0ABD- E9 0A             SBC #10
0ABF- C8                INY                 INC. QUOTIENT
0AC0- B0 F8             BCS .2              ...TRY ANOTHER SUBTRACTION
0AC2- 88                DEY                 CORRECT QUOTIENT
0AC3- 98                TYA                 GET QUOTIENT
0AC4- 0A        .3      ASL                 LEFT JUSTIFY
0AC5- 0A                ASL
0AC6- 0A                ASL
0AC7- 0A                ASL
0AC8- 0D 0D 08          ORA ARG.HI          MERGE WITH NEXT DIGIT
0ACB- 8D 0D 08          STA ARG.HI
0ACE- 8E 0C 08          STX ARG.EXPONENT    $41 OR $42
0AD1- A2 09             LDX #9              CLEAR REST OF ARG
0AD3- A9 00             LDA #0
0AD5- 9D 0D 08  .4      STA ARG.HI,X
0AD8- CA                DEX
0AD9- D0 FA             BNE .4
0ADB- 20 FF FF          JSR DADD
                *---SUBTRACT 0.5----------------
0ADE- A9 25     .5      LDA #CON.1HALF
0AE0- A0 08             LDY /CON.1HALF
0AE2- 20 FF FF          JSR MOVE.YA.ARG.1
0AE5- A9 FF             LDA #$FF
0AE7- 8D 17 08          STA ARG.SIGN
0AEA- 4C FF FF          JMP DADD
```

S-C Macro Assembler Version 2.0....................Bill Morgan

We are now accepting orders for the upgrade to S-C Macro Assembler Version 2.0. Here is a summary of the new features:

o The big news, of course, is the ability to assemble 65C02, 65802, and 65816 opcodes. The new .OP directive switches between the 6502, Sweet-16, 65C02, and 65816 opcode sets.

o All screen output now passes through one driver routine, which will be much easier to modify for other displays. Drivers are included for 40-column, //e and //c 80-column, and STB-80.

o Typing a Control-C at the command prompt (:) emits CATALOG, leaving the cursor at the end of the line, to add slot and drive specifiers if needed.

o There is a sort of Auto-SAVE function. Once you have created a comment line near the beginning of your source file containing the phrase "SAVE filename", typing ESC-S will emit that phrase and position the cursor at the end, so you can add a suffix or just press RETURN.

o The COPY command asks "DELETE ORIGINAL?" If you type "Y", the effect will be that of a MOVE command.

o The tape LOAD and SAVE commands have been removed, to make room for new features.

o All operand expressions are calculated to 32 bits and .DA data values may be larger, to allow for the 65816's extended addressing capabilities.

o You can force Zero Page or Absolute addressing modes by prefixing the operand with < or >.

o Operand expressions may include bitwise logical operations. &, ! (or |), and ^ are AND, OR, and EOR.

o Control-S functions as a case lock key, toggling upper/lower case entry.

o The .BS directive allows you to specify the value of the fill byte generated. This directive now creates fill bytes in assemblies into memory, rather than to disk only.

o The assembler tests for the "/" command character, to simplify use of the Laumer Research Full Screen Editor.

o All object code bytes are vectored through a standard location, so you can intercept the assembler's output for special purposes.

Registered owners of S-C Macro Assembler will be able to purchase the upgrade to Version 2.0 for only $20.00. Just send us a check or charge card number, and you will be among the first to have the new version.

Convert Two Decimal Digits to Binary.......Bob Sander-Cederlof

I have recently been running into more and more uses for the decimal mode in the 6502. In the decimal mode, each byte contains a value from 0 to 99, with the ten's digit in the left nybble and the units digit in the right nybble.

The 6502 has built-in capability to add and subtract values in this format, with automatic carry when a nybble exceeds 9. If you have been following my series on 18-digit arithmetic, you have seen a lot of examples of its use.

A frequent problem that arises is conversion between the decimal form and the binary form of a number. I suppose I have written ten million different programs to do this kind of conversion, on at least a thousand different kinds of computers! (Ever notice that my exaggerations are always in decimal?)

For a small (byte-size) example, suppose a byte contains two decimal digits ($00-$99) and you want to convert it to binary ($00-$63). The first step is to separate the two digits into two different variables. The the ten's digit should be multiplied by ten, and the unit's digit added.

Lines 1390-1510 in the listing perform these steps, but there are a few tricks. Lines 1410-1420 strip out the unit's digit and save it in LOW, and lines 1440-1450 save the high digit in HIGH. Notice that I did not shift the high digit down, so it is really the ten's digit times 16 (call it "tens*16").

Lines 1460-1500 multiply the tens*16 by 10/16. Then line 1500 adds the unit's digit.

The program in lines 1010-1190 is a test driver, which calls the DEC.HEX.2 routine 100 times with successive values in the A-register between $00 and $99. DEC.HEX.2 returns with the converted value ($00-$63 in the A-register, and the test driver prints out the value. If everything is okay, the hexadecimal numbers from $00 through $63 will be displayed.

DEC.HEX.2 as written takes 18 bytes plus two variables in page zero. If the variables are not in page zero, the program will take an additional four bytes.

A faster program which takes only a few more bytes, and does not use any variables in RAM other than the stack, is shown in lines 1200-1340. Lines 1220-1260 convert the ten's digit into an index 0-9 in the X-register. Line 1270 retrieves the original number from the stack. Lines 1290-1300 add a value from the table, indexed by the ten's digit, giving a total which is the converted number.

The values in the table consist of one byte each, having selected so that they subtract out the hexadecimal value of the ten's digit and add back the value of that digit-times-ten in binary. For example, if the original number was $58 (meaning decimal 58 in BCD storage format), we will add the value $E2

(which is 50-$50). $58+$E2 = $3A, which is the correct hexadecimal conversion.

I recently worked on a consulting project which included a lot of mixed decimal and hexadecimal calculations. The project was implemented on a 6511 chip, which has only 192 bytes of RAM. That is total, including the stack! We also had 4096 bytes of EPROM. The system operates in a real-time mode with relatively high-speed interrupts occurring. With these constraints, every routine had to be written to use the minimum amount of RAM and to be as fast as possible. A few extra bytes of code would be all right, because 4096 bytes of EPROM was more than enough. In situations like this, programs like the one in lines 1200-1300 come in real handy.

```
                    1000 *SAVE S.QUICK DEC-HEX
                    1010 *-------------------------------
0800- A9 00         1020 T       LDA #0
0802- 85 00         1030         STA 0
0804- A5 00         1040 .1      LDA 0
0806- 20 38 08      1050         JSR DEC.HEX.2
0809- 20 DA FD      1060         JSR $FDDA
080C- A9 A0         1070         LDA #" "
080E- 20 ED FD      1080         JSR $FDED
0811- 20 ED FD      1090         JSR $FDED
0814- F8            1100         SED
0815- 18            1110         CLC
0816- A5 00         1120         LDA 0
0818- 69 01         1130         ADC #1
081A- 85 00         1140         STA 0
081C- D8            1150         CLD
081D- C9 00         1160         CMP #0
081F- D0 E3         1170         BNE .1
0821- 60            1180         RTS
                    1190 *-------------------------------
                    1200 DEC.HEX
0822- 48            1210         PHA           SAVE BYTE
0823- 4A            1220         LSR
0824- 4A            1230         LSR
0825- 4A            1240         LSR
0826- 4A            1250         LSR
0827- AA            1260         TAX           HI NYBBLE TO X
0828- 68            1270         PLA           GET ORIG BYTE
0829- 18            1280         CLC
082A- 7D 2E 08      1290         ADC TBL,X
082D- 60            1300         RTS
                    1310 *-------------------------------
082E- 00 FA F4
0831- EE            1320 TBL     .DA #0-0,#10-$10,#20-$20,#30-$30
0832- E8 E2 DC      1330         .DA #40-$40,#50-$50,#60-$60
0835- D6 D0 CA      1340         .DA #70-$70,#80-$80,#90-$90
                    1350 *-------------------------------
01-                 1360 LOW     .EQ 1
02-                 1370 HIGH    .EQ 2
                    1380 *-------------------------------
                    1390 DEC.HEX.2
0838- 48            1400         PHA
0839- 29 0F         1410         AND #$0F      SAVE LOW NYBBLE
083B- 85 01         1420         STA LOW
083D- 68            1430         PLA
083E- 29 F0         1440         AND #$F0      GET HIGH NYBBLE
0840- 85 02         1450         STA HIGH
0842- 4A            1460         LSR           /2
0843- 4A            1470         LSR           /4
0844- 65 02         1480         ADC HIGH      /4*5
0846- 4A            1490         LSR           /8*5 = *10/16
0847- 65 01         1500         ADC LOW       + LOW NYBBLE
0849- 60            1510         RTS
                    1520 *-------------------------------
```

A Whole Megabyte for your Apple //e........ Bob Sander-Cederlof

Both Applied Engineering and Saturn have announced 1 Mbyte cards for the //e. Saturn's, I understand, plugs into any slot 1-7; this of course makes it a little non-standard compared to other //e memory expanders when it comes to software access.

The new board from Applied Engineering, called RAM WORKS, fits in the //e auxiliary slot. You get 80 column text and double hi-res, with anywhere from 64K to 1 Megabyte of expansion RAM in 64K or 256K increments. You can buy RAM WORKS already expanded, or expand it yourself later. Prices: 64K = $179, 128K = $249, 256K = $449, 512K = $799, and 1Meg = $1499. The first 512K fits one a normal size card, about 6 inches long. The second 512K come in a piggy-back card which attaches to the main card. Another option, an RGB video generator ($129), attaches to the front of the memory card.

The megabyte is divided into 16 chapters of 64K each. You select which one is active by storing a value from $00 to $0F in a register at $C073. Then the normal //e maze of soft switches lets you access the active chapter the same way you would access Apple's standard 64K card.

RAM WORKS has some new design ideas, for which patents are pending, including a power saving circuit and a video refresh circuit. The latter eliminates the annoying screen flicker that normally occurs when you switch chapters with older expansion cards.

Low cost software options available with RAM WORKS include disk emulation for DOS and ProDOS, and workspace expansion for Appleworks. Standard ProDOS will turn Apple's RAM card into a half-size RAMdisk, but with RAM WORKS you get a full megabyte!

If you like the idea of souping up your //e, one of these boards plus a new 65802 processor may be just the ticket!

65816 News..Bill Morgan

Did you see the Infoworld article a few weeks ago (November 5 issue) about the 65816? That story mentioned a plug-in board for the Apple II containing a 65816 processor and extra RAM. Well, I spoke today with Larry Hittel of Com Log, producers of that board, and it does sound very interesting.

Com Log intended their board, the Apple16, to be a developers' tool, rather than a consumer item, or an Apple hot-rod device. They were therefore a little surprised and overwhelmed by the response to the Infoworld story: When I talked to Larry they had exactly one board in stock, and it was waiting for purchase order paperwork from Apple Computer. They are a month or two away from full production quantities.

The Apple16 board uses DMA (Direct Memory Access) to take control of the Apple, shutting down the 6502 and taking over the address bus. They have found that the DMA does not function properly in Apples earlier than Revision 4, due to problems with the bus driver chips on the motherboard.

The 65816 chips are designed to operate at 8 MHz and are currently testing out at 2-4 MHz, but, in order to maintain compatibility with the Apple, the Com Log processor is clocked at 1 MHz.

To the '816, the 64K of Apple memory, both RAM and ROM, is bank 0. Bank 1 echoes the Apple from 0-DFFF, but contains space for new EPROM at E000-FFFF. Banks 2 and 3 are reserved for more new EPROM. Banks 4-7 are the on-board RAM, consisting of one set of either 64K or 256K chips. Banks 8-255 are available on an expansion connector, intended for a future separate memory board. There is abort logic to provide an interrupt on access to non-existent memory.

Com Log is selling the boards now with no EPROMs. They are working on an operating system and an Applesoft interpreter, but those are still some time away. No price has been set for the firmware yet.

The current price of the Apple16 board is $395 with no RAM, $450 with 64K, and $795 with 256K. They are not expecting to have them available in production quantities until January or later, by which time the prices might change. Contact Com Log Corporation at 11056 N. 23rd Dr., Suite 104, Phoenix, AZ 85029. Phone (602) 248-0769.

That Infoworld story quoted an Apple spokesman as saying that the 65816 was to be used in an earlier project that had been shelved. That project is being dusted off and revived, now that the 65816 chips are really coming through. We've been hearing of it as the Apple //x. According to an article in the November 19 issue of Infoworld about an interview with Woz, the //x is still not a fixed design and will not be ready for market until 1986. There's always something new to look forward to!

New DP18 Square Root Subroutine...........Bob Sander-Cederlof

Even after bending over backwards to be certain I had the best possible SQR implementation in the October AAL, I still found some ways to improve it. Last night I found some more information in a book called "Software Manual for the Elementary Functions", by William Cody and William Waite, Prentice-Hall, 1980.

They pointed out that in general an extra Newton iteration took less time than a complex method of getting an initial approximation which would be accurate enough to avoid one iteration. In other words, using a cubic polynomial like I did in October is just not worth it. Not worth the time, and not worth the space.

They further pointed out that it is best to compute the last Newton iteration in a slightly different fashion, to avoid shifting out the last significant digit. The normal iteration computes (x/y + y)*.5. Re-arrangement to y+(x/y-y)*.5 is better. Since it takes an extra step, it should only be used the last time.

To see the difference, consider the example below. I have used a precision of just 3 digits (instead of 18 or 20)to simplify the illustration:

```
let x=.253, and y=.5
then x/y=.506

x/y+y=1.00 (truncating to 3 places)
(x/y+y)*.5 = .500, which is wrong

x/y-y=.006
(x/y-y)*.5=.003
y+(x/y-y)*.5 = .503, which is correct.
```

My new SQR version uses a much faster method for getting the first approximation. The first two digits of the argument (in DAC.HI) must be in the range from 10 to 99. I convert them to an index between $02 and $13 by shifting the first digit over three, and adding one if the second digit is 5 or more. In other words, 10-14 become $02, $15-19 become $03, on up to $95-99 becoming $13. Then I use that value as an index into a table which gives a good approximation to the first two digits of the square root. For example, any number between .10 and .19999...9 will get a first approximation of .35. I store those two digits into DAC.HI, letting the remaining digits stay as they were. This method gives a first approximation which in the worst case still has at least the first digit correct.

It turns out the worst case is for numbers with odd exponents and the mantissa=1, such as 1 (which is .1*10^1), 100 (which is .1*10^3), and so on. Even in this worst case, four iterations give 20 digits of precision.

The end result of these changes is a faster and shorter program which is more accurate. Here is the new listing:

```
1000 *SAVE S.NEW SQR ROUTINE
1010 *-------------------------------
1020 *       SQR (DAC)
1030 *-------------------------------
1040 ERR.SQ JMP AS.ILLERR  ILLEGAL QUANTITY
1050 DP.SQR.0 RTS
1060 DP.SQR LDA DAC.EXPONENT
1070        BEQ DP.SQR.0  SQR(0)=0
1080        LDA DAC.SIGN
1090        BMI ERR.SQ    MUST BE POSITIVE
1100        JSR MOVE.DAC.TEMP3 SAVE X
1110 *---APPROX. ROOT OF .1 - 1-------
1120        LDA DAC.HI    CONVERT TWO DIGITS TO BINARY
1130        AND #$0F      SAVE LO DIGIT
1140        CMP #5        01234 OR 56789
1150        PHP           SAVE ANSWER
1160        LDA DAC.HI    GET HI DIGIT
1170        LSR
1180        LSR
1190        LSR
1200        LSR           $01...$09
1210        PLP           01234 OR 56789
1220        ROL           $02...$13
1230        TAX
1240        LDA SQR.TBL,X
1250        STA DAC.HI
1260 *---TAKE HALF OF EXPONENT--------
1270        LDA DAC.EXPONENT
1280        SEC
1290        SBC #$40      REMOVE OFFSET
1300        ROR           DIVIDE BY TWO (KEEP SIGN)
1310        PHP           SAVE ODD/EVEN BIT
1320        CLC
1330        ADC #$C0      RE-BIAS EXPONENT
1340        STA DAC.EXPONENT
1350        PLP
1360        BCC .1        EVEN, DON'T MULT BY SQR(10)
1370 *---ADJUST APPROX FOR ODD EXP----
1380        LDA #CON.SQR10
1390        LDY /CON.SQR10
1400        JSR MOVE.YA.ARG.1
1410        JSR DMULT
1420 *---THREE NEWTON ITERATIONS------
1430 .1     LDA #3
1440        STA TEMP3
1450 .2     JSR MOVE.DAC.TEMP2     TEMP2 = Y
1460        JSR MOVE.TEMP3.ARG     GET X
1470        JSR DDIV               X/Y
1480        JSR MOVE.TEMP2.ARG
1490        JSR DADD               X/Y+Y
1500        LDA #CON.HALF
1510        LDY /CON.HALF
1520        JSR MOVE.YA.ARG.1
1530        JSR DMULT              (X/Y+Y)/2
1540        DEC TEMP3              ANY MORE?
1550        BNE .2                 ...YES
1560 *---ONE MORE NEWTON ITERATION----
1570        JSR MOVE.DAC.TEMP2     TEMP2 = Y
1580        JSR MOVE.TEMP3.ARG     GET X
1590        JSR DDIV               X/Y
1600        JSR MOVE.TEMP2.ARG
1610        LDA #$FF
1620        STA ARG.SIGN
1630        JSR DADD               X/Y-Y
1640        LDA #CON.HALF
1650        LDY /CON.HALF
1660        JSR MOVE.YA.ARG.1
1670        JSR DMULT              (X/Y-Y)/2
1680        JSR MOVE.TEMP2.ARG
1690        JMP DADD               Y + (X/Y-Y)/2
1700 *-------------------------------
1710 SQR.TBL     .EQ *-2  (NO ENTRIES AT 0...1)
1720             .HS 35.42.47.52.57.61.65.69.72
1730             .HS 76.79.82.85.88.91.94.96.99
1740 CON.SQR10   .HS 41316227766016837933 20
1750 CON.HALF    .HS 40500000000000000000 00
1760 *-------------------------------
```

Improvements to 80-column Monitor Dump.........Jan Eugenides

I found a little bug in the 80-column ASCII monitor dump, as presented in Sept 1983 AAL (page 27,28). It worked great in the 80-column mode, but if I happened to be in 40-column mode when I used the monitor dump command something strange happens.

Some time ago I incorporated the dump and Steve Knouse's monitor patches into an EPROM and installed it in my system. Everything seemed to be working fine, until one day.... I was working on a short Applesoft program, and I went into the monitor in 40-column mode to check a few program bytes. When I returned to Applesoft and listed the program, the first line had been changed. Huh?

I eventually figured out that the problem had to do with the tab to column 60. In 40-column mode this will be 20 characters beyond the bottom of the screen, which is $80C.

The solution was to just print a few spaces rather than attempting to tab. This approach makes for more compatibility among various 80-column devices, too.

While I was at it, I even squeezed a byte out of the code.

[And I squeezed some more, saving a total of 11 bytes. Bob S-C]

Here is the modified routine:

```
                  1000 *SAVE S.NEW 80 COL MONITOR DUMP
                  1010 *--------------------------------
                  1020 *    TO INSTALL,
                  1030 *       1.  ASSEMBLE THIS PROGRAM
                  1040 *       2.  ENTER THESE MONITOR COMMANDS
                  1050 *      $C083 C083 FCC9<CC9.CFFM
                  1060 *      $FDBE:C9 FC N FDA6:F N FDB0:F
                  1070 *--------------------------------
                  1080 *   BY JAN EUGENIDES & BOB S-C
                  1090 *--------------------------------
24-               1100 CH     .EQ $24
3C-               1110 A1     .EQ $3C,3D
3E-               1120 A2     .EQ $3E,3F
42-               1130 A4     .EQ $42,43
02F0-             1140 BUFFER .EQ $2F0
FDDA-             1150 PRBYTE .EQ $FDDA
FDED-             1160 COUT   .EQ $FDED
F948-             1170 PRBLNK .EQ $F948
                  1180 *--------------------------------
                  1190        .OR $FCC9
                  1200        .TA $CC9
                  1210 *--------------------------------
FCC9- 48          1220 PATCH  PHA             SAVE BYTE
FCCA- A5 3C       1230        LDA A1          COMPUTE INDEX
FCCC- 29 0F       1240        AND #$0F        0...F
FCCE- AA          1250        TAX
FCCF- 68          1260        PLA             GET BYTE AGAIN
FCD0- 9D F0 02    1270        STA BUFFER,X    SAVE IN BUFFER
FCD3- 20 DA FD    1280        JSR PRBYTE      PRINT ON SCREEN
FCD6- E8          1290        INX             GET # BYTES THIS LINE
FCD7- 86 42       1300        STX A4          SAVE IN A4L
FCD9- E0 10       1310        CPX #$10        END OF LINE?
FCDB- F0 0A       1320        BEQ .1          ...YES, PRINT ASCII CHARS
FCDD- A5 3C       1330        LDA A1          ...NO, SEE IF END OF RANGE
FCDF- C5 3E       1340        CMP A2
FCE1- A5 3D       1350        LDA A1+1
FCE3- E5 3F       1360        SBC A2+1
FCE5- 90 18       1370        BCC .4          ...NO, RETURN
```

```
FCE7- 20 48 F9 1380 .1      JSR PRBLNK   PRINT 3 SPACES
FCEA- A2 00     1390        LDX #0       PRINT ASCII CHARS FROM BUFFER
FCEC- BD F0 02 1400 .2      LDA BUFFER,X GET CHAR
FCEF- 09 80     1410        ORA #$80     MAKE NORMAL VIDEO
FCF1- C9 A0     1420        CMP #$A0     TRAP CONTROL CHARS
FCF3- B0 02     1430        BCS .3       ...NOT CONTROL CHAR
FCF5- A9 AE     1440        LDA #$AE     ...CTRL, SUBSTITUTE "."
FCF7- 20 ED FD 1450 .3      JSR COUT     PRINT CHAR
FCFA- E8        1460        INX          NEXT
FCFB- E4 42     1470        CPX A4       END OF LIST?
FCFD- 90 ED     1480        BCC .2       ...NOT YET
FCFF- 60        1490 .4     RTS          RETURN
```

Note the directions for installing the routine in a RAM card copy of the monitor, in lines 1020-1060. "$C083 C083 FCC9<CC9.CFFM" write enables the RAM area and copies the dump code over the top of cassette I/O stuff. "$FDBE:C9 FC N FDA6:F N FDB0:F" patches the monitor dump command code to call the new patch, and also patches to print 16 bytes per screen line.

If you want to use this routine in 40-column mode only, change line 1240 from "AND #$0F" to "AND #$07", line 1310 from "CPX #$10" to "CPX #$08", and leave out the patches at FDA6 and FDB0 in the previous paragraph.

Generating Cross Reference Text File with DISASM...Bob Kovacs

I received a phone call from Don Lancaster the other day. He had been using DISASM to probe the mysteries of AppleWriter, and was now preparing to document his findings. Although he liked the way DISASM generated a triple cross reference table, he preferred to have it in a form that could be used by his word processor (that is, on a text file). The cross reference table generated by DISASM is normally output to either the screen or a printer, so Don's only alternative was to manually type it into his word processor. There were hundreds of labels....

It turned out that a simple patch to DISASM will do the trick. All that is necessary is to change the JSR PASS2 which normally generates the source code listing to JSR XREF.

The following patch outputs the cross reference table to your file after responding "Y" to the prompt "GENERATE TEXT FILE?":

```
$09A1:20 F1 0A
```

Back in the April issue of AAL, I described a method of using EXEC files with DISASM. A patch was required to the "YES/NO" routine to input the response via KEYIN rather than directly from the keyboard. Although the patch I gave in April works, KEYIN uses the Y-register as an index to the screen. My patch did not always wind up in the right place. So I have expanded the patch as follows:

```
$0C57:EA A4 24 20 18 FD 09 80
```

I hope that this has not caused any inconvenience.

Macro Information by Example..................Sandy Greenfarb

The following are three examples of macro use which I have found interesting and informative.

The first example, TEST, shows that you can use parameters in places other than the operand field. In this case, one of the parameters becomes part of an opcode name.

SETD shows how a macro can make more efficient code. If both bytes are the same, there is no need to have two LDA instructions.

MOVD copies two bytes from one variable to another. If you use MOVD to move two bytes one byte higher in RAM, MOVD will reverse the order the bytes are moved so that the data are not clobbered.

```
                  1000 *SAVE S.MACRO EXAMPLES
                  1010 *--------------------------------
                  1020 *     BY SANDY GREENFARB
                  1030 *--------------------------------
                  1040 *
                  1050 *     PARAMETERS CAN SUBSTITUTE ANYWHERE,
                  1060 *       EVEN IN OPCODES
                  1070 *--------------------------------
                  1080        .MA TEST      VALUE,CONDITION,LABEL
                  1090        CMP ]1
                  1100        B]2 ]3
                  1110        .EM
                  1120 *
0800-             1130        >TEST #3,CC,SMALLER
0800- C9 03       0000>        CMP #3
0802- 90 07       0000>        BCC SMALLER
0804-             1140        >TEST TYPE,EQ,SAME
0804- CD 09 08    0000>        CMP TYPE
0807- F0 01       0000>        BEQ SAME
                  1150 *
0809- 23          1160 TYPE    .DA #35
080A- EA          1170 SAME    NOP
080B- EA          1180 SMALLER NOP
                  1190 *--------------------------------
                  1200 *
                  1210 *    MACROS CAN SIMPLIFY CODE FOR EFFICIENCY
                  1220 *--------------------------------
                  1230        .MA SETD      VALUE,VARIABLE
                  1240        LDA #]1       LO-BYTE
                  1250        STA ]2
                  1260     .DO ]1/256*257-]1  ARE LOW AND HI EQUAL?
                  1270        LDA /]1
                  1280     .ELSE
                  1290 *                    HI = LO-BYTE
                  1300     .FIN
                  1310        STA ]2+1
                  1320        .EM
                  1330 *
080C-             1340        >SETD $1234,VALUE
080C- A9 34       0000>        LDA #$1234       LO-BYTE
080E- 8D 1E 08    0000>        STA VALUE
                  0000>     .DO $1234/256*257-$1234  ARE LOW AND HI EQUAL?
0811- A9 12       0000>        LDA /$1234
                  0000>     .ELSE
                  0000>     .FIN
0813- 8D 1F 08    0000>        STA VALUE+1
0816-             1350        >SETD $2323,VALUE
0816- A9 23       0000>        LDA #$2323       LO-BYTE
0818- 8D 1E 08    0000>        STA VALUE
                  0000>     .DO $2323/256*257-$2323  ARE LOW AND HI EQUAL?
                  0000>     .ELSE
                  0000> *                   HI = LO-BYTE
                  0000>     .FIN
081B- 8D 1F 08    0000>        STA VALUE+1
                  1360 *
081E-             1370 VALUE  .BS 2
```

```
                 *-------------------------------
                 *
                 *   MACROS CAN PREVENT PROGRAMMING MISTAKES
                 *      SUCH AS OVER-WRITING WHEN YOU COPY
                 *      ONE VARIABLE INTO ANOTHER.
                 *-------------------------------
                        .MA MOVD      VAR1,VAR2
                     .DO ]2-]1-1
                        LDA ]1        NO OVERLAP
                        STA ]2
                        LDA ]1+1
                        STA ]2+1
                     .ELSE
                        LDA ]1+1      THIS CODE BUILT WHEN THE
                        STA ]2+1      VARIABLES OVERLAP
                        LDA ]1
                        STA ]2
                     .FIN
                        .EM
                 *
0820-                   >MOVD $11,$22
                      .DO $22-$11-1
0820- A5 11             LDA $11       NO OVERLAP
0822- 85 22             STA $22
0824- A5 12             LDA $11+1
0826- 85 23             STA $22+1
                      .ELSE
                      .FIN
0828-                   >MOVD $28,VALUE
                      .DO VALUE-$28-1
0828- A5 28             LDA $28       NO OVERLAP
082A- 8D 1E 08          STA VALUE
082D- A5 29             LDA $28+1
082F- 8D 1F 08          STA VALUE+1
                      .ELSE
                      .FIN
0832-                   >MOVD $11,$12
                      .DO $12-$11-1
                      .ELSE
0832- A5 12             LDA $11+1     THIS CODE BUILT WHEN THE
0834- 85 13             STA $12+1     VARIABLES OVERLAP
0836- A5 11             LDA $11
0838- 85 12             STA $12
                      .FIN
                 *-------------------------------
```

Turning Bit-Masks into Indices............Bob Sander-Cederlof

A few months ago I presented several ways to turn an index (0-7) into a bit mask (01, 02, 04,...,80). We got a lot of feedback, including some faster and better programs. Bruce Love suggested the possibility of the reverse transformation.

According to Bruce, who is a high school teacher in New Zealand, the method which uses the fewest bytes is the one I show in lines 1390-1450. In order to be fair in comparing different algorithms, I am going to count the RTS opcodes both for bytes and for cycles. With this in mind, Bruce's routine takes 8 bytes and from 16 to 65 cycles. This is certainly the smallest way, and it really is pretty fast.

Bruce mentioned that he had written several other programs to solve the same problem: one used the X-register, took 26 bytes with an average of 33.5 cycles; another without useing X or Y took 28 bytes and an average of 39 cycles. Unfortunately, he did not include a copy of either of these.

I worked out four more methods, shown in the listing after Bruce's. I wrote a test driver which is in lines 1000-1310. The test driver calls each routine, printing the results of each, for all possible values of the bit-mask.

The following table summarizes the data for the five algorithms:

		# of cycles		
	bytes	min	max	ave
SMALLEST.WAY	8	16	65	40.5
WAY.WITH.X	26	25	42	33.5
WAY.WITHOUT.X	23	14	30	22
ANOTHER.WAY.W...	32	14	24	18.375
STRAIGHT.TEST...	33	14	27	18.5

If the SMALLEST.WAY is not fast enough, I would probably go with the one named WAY.WITHOUT.X. It is almost as fast as the fastest, and is the shortest of the longer routines. Of course, some of you may come up with better and faster ones....

```
                    1000 *SAVE S.MASK --> INDEX
                    1010 *--------------------------------
0800- A0 01         1020 TEST     LDY #$01
0802- 98            1030 .1       TYA
0803- 20 DA FD      1040          JSR $FDDA
0806- 98            1050          TYA
0807- 20 3C 08      1060          JSR SMALLEST.WAY
080A- 20 32 08      1070          JSR HEX
080D- 98            1080          TYA
080E- 20 44 08      1090          JSR WAY.WITH.X
0811- 20 32 08      1100          JSR HEX
0814- 98            1110          TYA
0815- 20 5E 08      1120          JSR WAY.WITHOUT.X
0818- 20 32 08      1130          JSR HEX
081B- 98            1140          TYA
081C- 20 75 08      1150          JSR ANOTHER.WAY.WITHOUT.X
081F- 20 32 08      1160          JSR HEX
0822- 98            1170          TYA
0823- 20 95 08      1180          JSR STRAIGHT.TESTING.WAY
0826- 20 32 08      1190          JSR HEX
0829- 20 8E FD      1200          JSR $FD8E
```

```
082C- 98                  TYA
082D- 0A                  ASL
082E- A8                  TAY
082F- 90 D1               BCC .1
0831- 60                  RTS
                *--------------------------------
0832- 48        HEX       PHA
0833- A9 AD               LDA #"-"
0835- 20 ED FD            JSR $FDED
0838- 68                  PLA
0839- 4C DA FD            JMP $FDDA
                *--------------------------------
                *   WAY WITH FEWEST BYTES
                *       8 BYTES
                *       MIN:  16 CYCLES
                *       MAX:  65 CYCLES
                *       AVE:  40.5 CYCLES
                *--------------------------------
                SMALLEST.WAY
083C- A2 08               LDX #8
083E- CA        .1        DEX
083F- 0A                  ASL
0840- 90 FC               BCC .1
0842- 8A                  TXA
0843- 60                  RTS
                *--------------------------------
                *   FASTER WAY USING X-REGISTER
                *       26 BYTES
                *       MIN: 25 CYCLES
                *       MAX: 42 CYCLES
                *       AVE: 33.5 CYCLES
                *--------------------------------
                WAY.WITH.X
0844- A2 00               LDX #0        KEEP INDEX IN X
0846- C9 10               CMP #$10      80-40-20-10 / 08-04-02-01
0848- 90 06               BCC .1        ...8,4,2,1
084A- 4A                  LSR           ...80,40,20,10
084B- 4A                  LSR           SHIFT OVER TO 8,4,2,1
084C- 4A                  LSR
084D- 4A                  LSR
084E- A2 04               LDX #4        AND BUMP INDEX BY 4
0850- C9 04     .1        CMP #$04      08-04 / 02-01
0852- 90 04               BCC .2        ...2,1
0854- 4A                  LSR           ...8,4
0855- 4A                  LSR           SHIFT OVER TO 2,1
0856- E8                  INX           AND BUMP INDEX BY 2
0857- E8                  INX
0858- 4A        .2        LSR           02 / 01
0859- F0 01               BEQ .3        ...01
085B- E8                  INX           ...02, BUMP INDEX
085C- 8A        .3        TXA           GET RESULT
085D- 60                  RTS
                *--------------------------------
                *   WAY WITHOUT USING X-REGISTER
                *       23 BYTES
                *       MIN: 14 CYCLES
                *       MAX: 30 CYCLES
                *       AVE: 22 CYCLES
                *--------------------------------
                WAY.WITHOUT.X
085E- 4A                  LSR           40-20-10-08-04-02-01-00
085F- C9 04               CMP #$04
0861- 90 0E               BCC .2        ...2,1,0
0863- F0 0D               BEQ .3        ...4, SHOULD BE 3
0865- 4A                  LSR           20-10-08-04
0866- 4A                  LSR           10-08-04-02
0867- 4A                  LSR           08-04-02-01
0868- 4A                  LSR           04-02-01-00
0869- C9 04               CMP #4
086B- 90 02               BCC .1        2,1,0 INTO 6,5,4
086D- A9 02               LDA #2        4 INTO 7
086F- 69 04     .1        ADC #4
0871- 60        .2        RTS
0872- E9 01     .3        SBC #1        4 INTO 3
0874- 60                  RTS
```

```
                  1960 *-------------------------------
                  1970 *   ANOTHER WAY WITHOUT X-REGISTER
                  1980 *       32 BYTES
                  1990 *       MIN: 14 CYCLES
                  2000 *       MAX: 24 CYCLES
                  2010 *       AVE: 18.375 CYCLES
                  2020 *-------------------------------
                  2030 ANOTHER.WAY.WITHOUT.X
0875- C9 08       2040        CMP #$08       80-40-20-10-08-04-02-01
0877- 90 1A       2050        BCC .5         ...4,2,1
0879- F0 16       2060        BEQ .4         ...8, SHOULD BE 3
087B- C9 40       2070        CMP #$40
087D- 90 08       2080        BCC .2         ...20,10
087F- F0 03       2090        BEQ .1         ...40
0881- A9 07       2100        LDA #7
0883- 60          2110        RTS
0884- A9 06       2120 .1     LDA #6
0886- 60          2130        RTS
0887- C9 20       2140 .2     CMP #$20
0889- F0 03       2150        BEQ .3
088B- A9 04       2160        LDA #4
088D- 60          2170        RTS
088E- A9 05       2180 .3     LDA #5
0890- 60          2190        RTS
0891- E9 02       2200 .4     SBC #2
0893- 4A          2210 .5     LSR
0894- 60          2220        RTS
                  2230 *-------------------------------
                  2240 *   STRAIGHTFORWARD TESTING APPROACH
                  2250 *       33 BYTES
                  2260 *       MIN:  14 CYCLES
                  2270 *       MAX:  27 CYCLES
                  2280 *       AVE:  18.5 CYCLES
                  2290 *-------------------------------
                  2300 STRAIGHT.TESTING.WAY
0895- C9 08       2310        CMP #$08
0897- 90 1B       2320        BCC .5
0899- F0 16       2330        BEQ .4
089B- C9 20       2340        CMP #$20
089D- 90 0F       2350        BCC .3
089F- F0 0A       2360        BEQ .2
08A1- C9 80       2370        CMP #$80
08A3- 90 03       2380        BCC .1
08A5- A9 07       2390        LDA #7
08A7- 60          2400        RTS
08A8- A9 06       2410 .1     LDA #6
08AA- 60          2420        RTS
08AB- A9 05       2430 .2     LDA #5
08AD- 60          2440        RTS
08AE- A9 04       2450 .3     LDA #4
08B0- 60          2460        RTS
08B1- A9 03       2470 .4     LDA #3
08B3- 60          2480        RTS
08B4- 4A          2490 .5     LSR            CONVERT 4,2,1 TO 2,1,0
08B5- 60          2500        RTS
                  2510 *-------------------------------
```

Apple //e Reference Manual Source

We have located a mail- or phone-order source for the Apple manuals! A reader in New York City phoned to let us know that the McGraw-Hill Bookstore there carries the Apple publications. Apparently the bookstore is also a computer store and an Apple Dealer. The address is McGraw-Hill Bookstore, 1221 Sixth Ave., New York, NY 10020. The phone number is (212) 512-4100.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $12 postage for other countries. Back issues are available for $1.80 each (other countries add $1 per back issue for postage).